

ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE ASSL86
 OBJECT MODULE PLACED IN ASSL86.OBJ
 ASSEMBLER INVOKED BY: ASM86 ASSL86.ASM DEBUG

LOC	OBJ	LINE	SOURCE
		1	;
		2	;
		3	interfere module for it86
		4	6/15/81
		5	;
		6	group group dats
0000	3F	7	dats segment public 'DATS'
		8	db '?'
		9	dats ends
		10	;
		11	group group code
		12	extrn asmilinear
		13	public ddtgetline
		14	public goddt
		15	public ddtset
		16	;
		17	code segment public 'CODE'
		18	assume cs:group
0000		19	;
2300		20	assent proc
2300	E90000	21	org 2300h
		22	jmp asm
		23	assent endp
		24	;
2303		25	ddtgetline proc
2303	E903DE	26	jmp \$-21fah
		27	ddtgetline endp
		28	;
2306		29	goddt proc
2306	E903DE	30	jmp \$-21fah
		31	goddt endp
		32	;
2309		33	ddtset proc
2309	E903DE	34	jmp \$-21fah
		35	ddtset endp
		36	;
		37	code ends
			end

CP/M-86 v.1.1 (Vol. III)
 COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950
 SER. # C81-162

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
??SLG . . .	SEGMENT		SIZE=0000H PARA PUBLIC
ASM	L NEAR	0000H	EXTRN
ASSENT. . .	L NEAR	0000H	CODE
CGROUP. . .	GROUP		CODE
CODE. . . .	SEGMENT		SIZE=230CH PARA PUBLIC 'CODE'
DATS. . . .	SEGMENT		SIZE=0001H PARA PUBLIC 'DATS'
DDTGETLINE. L	NEAR	2303H	CODE PUBLIC
DDTSET. . .	L NEAR	2309H	CODE PUBLIC
DGROUP. . .	GROUP		DATS
GDDOT . . .	L NEAR	2306H	CODE PUBLIC

ASSEMBLY COMPLETE, NO ERRORS FOUND

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

!
!create ddt86 assembler and disassembler modules on VAX 780
!
$ pli gentab
$ link gentab
$ run gentab
! gentab reads op86.dat and creates optab.dat and optab.lit
!
$ plm86 ins86.plm debug pagewidth(100) xref
$ plm36 dis86.plm debug pagewidth(100) xref
$ asm86 disl86.asm debug
$ link86 disl86.obj, dis86.obj, ins86.obj to dis86.lnk
$ loc86 dis86.lnk to dis86.abs ad(sm(dats(0),code(0h))) -
  od(sm(dats,code,const,data))
$ h86 dis86.abs
!
$ plm86 ass86.plm debug xref pagewidth(100)
$ plm36 asmtab.plm debug xref pagewidth(100)
$ asm86 assl86.asm debug
$ link36 assl86.obj, asmtab.obj, ass86.obj, -
  po(dis86.abs) to ass86.lnk
$ loc36 ass86.lnk to ass36.abs ad(sm(dats(0),code(0h))) -
  od(sm(dats,code,const,data))
$ h86 ass36.abs
!
! now upload ass86.h86 and dis86.h86 to micro to link with ddt86
!

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE ASMTAB

OBJECT MODULE PLACED IN ASMTAB.OBJ

COMPILER INVOKED BY: PLM86 ASMTAB.PLN DEBUG XREF PAGEWIDTH(100)

```

$title('tables for ddt86 assembler')
$date(6/15/81)

```

```

1      asmtab: do;

```

```

      $nolist

```

```

4      1      declare op$tab (*) byte public data (

```

```

      aa$in, 1, 037h, 0,
      aad$in, 2, 0d5h, 0ah,
      aam$in, 2, 0d4h, 0ah,
      aas$in, 1, 03fh, 0,
      adc$in, 9, 10h, 2,
      add$in, 9, 0, 0,
      and$in, 10, 20h, 4,
      call$in, 11, 0, 0,
      callf$in, 17, 0, 0,
      cbw$in, 1, 098h, 0,
      clc$in, 1, 0f8h, 0,
      cld$in, 1, 0fch, 0,
      cli$in, 1, 0fah, 0,
      cmc$in, 1, 0f5h, 0,
      cmp$in, 9, 38h, 7,
      cpsb$in, 1, 0a6h, 0,
      cpsw$in, 1, 0a7h, 0,
      cwd$in, 1, 099h, 0,
      daa$in, 1, 027h, 0,
      das$in, 1, 02fh, 0,
      dec$in, 7, 0, 0,
      div$in, 5, 0f6h, 6,
      esc$in, 13, 0d8h, 0,
      hlt$in, 1, 0f4h, 0,
      idiv$in, 5, 0f6h, 7,
      imul$in, 5, 0f6h, 5,
      int$in, 15, 0e4h, 0,
      inct$in, 7, 1, 0,
      int$in, 12, 0, 0,
      into$in, 1, 0ceh, 0,
      iret$in, 1, 0cfh, 0,
      jaa$in, 3, 077h, 0,
      jaeb$in, 3, 073h, 0,
      jbb$in, 3, 072h, 0,
      jbe$in, 3, 076h, 0,
      jcb$in, 3, 072h, 0,
      jcxz$in, 3, 0e3h, 0,
      jeb$in, 3, 074h, 0,
      jg$in, 3, 07fh, 0,
      jge$in, 3, 07dh, 0,
      jlt$in, 3, 07ch, 0,
      jle$in, 3, 07eh, 0,
      jmp$in, 11, 1, 0,

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

japf\$in, 17, 1, 0,
jap\$in, 3, 0ebh, 0,
jna\$in, 3, 076h, 0,
jnae\$in, 3, 072h, 0,
jnb\$in, 3, 073h, 0,
jnbe\$in, 3, 077h, 0,
jnc\$in, 3, 073h, 0,
jne\$in, 3, 075h, 0,
jng\$in, 3, 07eh, 0,
jnge\$in, 3, 07ch, 0,
jnl\$in, 3, 07dh, 0,
jnle\$in, 3, 07fh, 0,
jno\$in, 3, 071h, 0,
jnp\$in, 3, 07bh, 0,
jns\$in, 3, 079h, 0,
jnz\$in, 3, 075h, 0,
jo\$in, 3, 070h, 0,
jp\$in, 3, 07ah, 0,
jpe\$in, 3, 07ah, 0,
jpo\$in, 3, 07bh, 0,
js\$in, 3, 078h, 0,
jz\$in, 3, 074h, 0,
lahf\$in, 1, 09fh, 0,
ld\$in, 6, 0c5h, 0,
lea\$in, 6, 08dh, 0,
lest\$in, 6, 0c4h, 0,
lock\$in, 0ffh, 0f0h, 0,
lodsb\$in, 1, 0ach, 0,
lods\$in, 1, 0adh, 0,
loop\$in, 3, 0e2h, 0,
loope\$in, 3, 0elh, 0,
loopne\$in, 3, 0e0h, 0,
loopnz\$in, 3, 0e0h, 0,
loopz\$in, 3, 0elh, 0,
mov\$in, 20, 0, 0,
movsb\$in, 1, 0a4h, 0,
movsw\$in, 1, 0a5h, 0,
mul\$in, 5, 0f6h, 4,
neg\$in, 5, 0f6h, 3,
nop\$in, 1, 090h, 0,
not\$in, 5, 0f6h, 2,
or\$in, 10, 3, 1,
out\$in, 16, 0e6h, 0,
pop\$in, 21, 0, 0,
popf\$in, 1, 09dh, 0,
push\$in, 3, 1, 0,
pushf\$in, 1, 09ch, 0,
rcl\$in, 4, 0d0h, 2,
rcr\$in, 4, 0d0h, 3,
rept\$in, 0feh, 0f3h, 0,
repe\$in, 0feh, 0f3h, 0,
repne\$in, 0feh, 0f2h, 0,
repnz\$in, 0feh, 0f2h, 0,
repz\$in, 0feh, 0f3h, 0,
ret\$in, 14, 0c2h, 0,
retf\$in, 14, 0cah, 0,
rol\$in, 4, 0d0h, 0,

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```
rort$in, 4, 0d0h, 1,  
sahf$in, 1, 09eh, 0,  
sal$in, 4, 0d0h, 4,  
sart$in, 4, 0d0h, 7,  
sbb$in, 9, 18h, 3,  
scasb$in, 1, 0aeh, 0,  
scasw$in, 1, 0afh, 0,  
shl$in, 4, 0d0h, 4,  
shr$in, 4, 0d0h, 5,  
stc$in, 1, 0f9h, 0,  
std$in, 1, 0fdh, 0,  
sti$in, 1, 0fbh, 0,  
stosb$in, 1, 0aah, 0,  
stosw$in, 1, 0abh, 0,  
sub$in, 9, 28h, 5,  
test$in, 18, 0, 0,  
wait$in, 1, 09bh, 0,  
xchg$in, 19, 0, 0,  
xlat$in, 1, 0d7h, 0,  
xor$in, 10, 30h, 6  
);
```

5 1

end asatlab;

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

CROSS-REFERENCE LISTING

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES
3			AABIN. LITERALLY 4
3			AADIN. LITERALLY 4
3			AAMIN. LITERALLY 4
3			AASIN. LITERALLY 4
3			ADCIN. LITERALLY 4
3			ADDIN. LITERALLY 4
3			ANDIN. LITERALLY 4
1	0000H		ASMTAB PROCEDURE STACK=0000H
3			CALLFIN. LITERALLY 4
3			CALLIN. LITERALLY 4
3			CBWIN. LITERALLY 4
3			CLCIN. LITERALLY 4
3			CLDIN. LITERALLY 4
3			CLIIN. LITERALLY 4
3			CNCIN. LITERALLY 4
3			CNPIN. LITERALLY 4
3			CNPSBIN. LITERALLY 4
3			CNPSWIN. LITERALLY 4
3			CSIN LITERALLY 4
3			CWDIN. LITERALLY 4
3			DABIN. LITERALLY 4
3			DASIN. LITERALLY 4
3			DECIN. LITERALLY 4
3			DIVIN. LITERALLY 4
3			DSIN LITERALLY 4
3			ESBIN. LITERALLY 4
3			ESIN LITERALLY 4
3			HLTIN. LITERALLY 4
3			IDIVIN. LITERALLY 4
3			IMULIN. LITERALLY 4
3			INCIN. LITERALLY 4
3			ININ LITERALLY 4
3			INTIN. LITERALLY 4
3			INTOIN. LITERALLY 4
3			IRETIN. LITERALLY 4
3			JAEIN. LITERALLY 4
3			JAIN LITERALLY 4
3			JBEIN. LITERALLY 4
3			JBIN LITERALLY 4
3			JCIN LITERALLY 4
3			JCXZIN. LITERALLY 4
3			JEIN LITERALLY 4
3			JGEIN. LITERALLY 4
3			JGIN LITERALLY 4
3			JLEIN. LITERALLY 4
3			JLIN LITERALLY 4
3			JMPFIN. LITERALLY 4
3			JMPIN. LITERALLY 4
3			JNPSIN. LITERALLY 4

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

3	JNAEIN	LITERALLY	4
3	JNAIN	LITERALLY	4
3	JNBEIN	LITERALLY	4
3	JNBIN	LITERALLY	4
3	JNCIN	LITERALLY	4
3	JNEIN	LITERALLY	4
3	JNGEIN	LITERALLY	4
3	JNGIN	LITERALLY	4
3	JNLEIN	LITERALLY	4
3	JNLIN	LITERALLY	4
3	JNOIN	LITERALLY	4
3	JNPIN	LITERALLY	4
3	JNSIN	LITERALLY	4
3	JNZIN	LITERALLY	4
3	JOIN	LITERALLY	4
3	JPEIN	LITERALLY	4
3	JPIN	LITERALLY	4
3	JPOIN	LITERALLY	4
3	JSIN	LITERALLY	4
3	JZIN	LITERALLY	4
3	LAHFIN	LITERALLY	4
3	LDSIN	LITERALLY	4
3	LEAIN	LITERALLY	4
3	LESIN	LITERALLY	4
3	LOCKIN	LITERALLY	4
3	LODSBIN	LITERALLY	4
3	LODSWIN	LITERALLY	4
3	LOOPEIN	LITERALLY	4
3	LOOPIN	LITERALLY	4
3	LOOPNEIN	LITERALLY	4
3	LOOPNZIN	LITERALLY	4
3	LOOPZIN	LITERALLY	4
3	MOVIN	LITERALLY	4
3	MOVSBIN	LITERALLY	4
3	MOVSWIN	LITERALLY	4
3	MULIN	LITERALLY	4
3	NEGIN	LITERALLY	4
3	NOPIN	LITERALLY	4
3	NOTIN	LITERALLY	4
2	OP2IN	LITERALLY	4
2	OP3IN	LITERALLY	4
2	OP4IN	LITERALLY	4
2	OP5IN	LITERALLY	4
2	OP6IN	LITERALLY	4
4	0000H 480 OPTAB	BYTE ARRAY(480) PUBLIC DATA	
3	ORIN	LITERALLY	4
3	OUTIN	LITERALLY	4
3	POPFIN	LITERALLY	4
3	POPIN	LITERALLY	4
3	PUSHFIN	LITERALLY	4
3	PUSHIN	LITERALLY	4
3	RCLIN	LITERALLY	4
3	RCRIN	LITERALLY	4
3	REPEIN	LITERALLY	4
3	REPIN	LITERALLY	4
3	REPNEIN	LITERALLY	4
3	REPNZIN	LITERALLY	4

COPYRIGHT © 1981, 1982
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

3	REPZIN	LITERALLY	4
3	RETFIN	LITERALLY	4
3	RETIN	LITERALLY	4
3	ROLIN	LITERALLY	4
3	RORIN	LITERALLY	4
3	SAHFIN	LITERALLY	4
3	SALIN	LITERALLY	4
3	SARIN	LITERALLY	4
3	SBBIN	LITERALLY	4
3	SCASBIN	LITERALLY	4
3	SCASWIN	LITERALLY	4
3	SHLIN	LITERALLY	4
3	SHRIN	LITERALLY	4
3	SSIN	LITERALLY	4
3	STCIN	LITERALLY	4
3	STDIN	LITERALLY	4
3	STIIN	LITERALLY	4
3	STOSBIN	LITERALLY	4
3	STOSWIN	LITERALLY	4
3	SUBIN	LITERALLY	4
3	TESTIN	LITERALLY	4
3	WAITIN	LITERALLY	4
3	XCHGIN	LITERALLY	4
3	XLATIN	LITERALLY	4
3	XORIN	LITERALLY	4

MODULE INFORMATION:

CODE AREA SIZE = 0000H 0D
 CONSTANT AREA SIZE = 01E0H 480D
 VARIABLE AREA SIZE = 0000H 0D
 MAXIMUM STACK SIZE = 0000H 0D
 268 LINES READ
 0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

ISIS-II PL/M-86 V2.0 COMPILATION OF MODULE DDTASM
 OBJECT MODULE PLACED IN ASS86.OBJ
 COMPILER INVOKED BY: PLM86 ASS86.PLN DEBUG XREF PAGERWIDTH(100)

```

$title('assembler for ddt86')
$date(6/15/81)
$compact
$optimize(2)

1      ddtasm: do;

2  1    declare
        logical literally 'byte',
        true literally '1',
        false literally '0',
        tab literally '09h',
        cr literally '0dh',
        lf literally '0ah';

3  1    declare
        tab$ptrs (5) address external,
        nops (5) byte external,
        opn$in (6) byte external,
        optab (480) byte external;

4  1    declare
        token (32) byte,          /* ASCII token */
        token$word address at (.token), /* used for word compares */
        token$len byte,          /* number of chars in token */
        token$type byte,         /* number, string, or delimiter */
        string$type literally '0',
        number$type literally '1',
        delim$type literally '2',
        token$value address,      /* value if token$type = number */
        next$ch byte;            /* next char of input stream */

5  1    declare
        assem$ptr pointer,
        assem$offset address at (.assem$ptr),
        assem$segment address at (.assem$ptr+2),
        assem$byte based assem$ptr byte;

6  1    declare
        seg$reg (8) byte EXTERNAL,
        reg16 (16) byte EXTERNAL,
        reg8 (16) byte EXTERNAL,
        base$reg (*) byte data ('BX', 'BP'),
        index$reg (*) byte data ('SI', 'DI');

7  1    declare
        op$base address,
        (op1, op2, op based op$base) structure (
            seg$reg$num byte,
            reg$num byte,
            base$reg$num byte,

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950
 SER. # _____

```

index$reg$num byte,
seg$reg$type logical,
reg$16$type logical,
reg$8$type logical,
num$type logical,
acc$type logical,
null$type logical,
far$type logical,
mod$rm$type logical,
w$bit byte,
bw$prefix byte,
sub$type byte,
seg$value address,
value address);

```

/*

op.subtypes

```

1  reg8
2  reg16
3  [num]
4  [pointer reg]
5  [index reg]
6  [pointer reg + index reg]
7  num [pointer reg]
8  num [index reg]
9  num [pointer reg + index reg]

```

*/

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

8  1  declare
      prefix (3) byte,      /* up to 3 prefixes */
      nprefix byte,        /* number of prefixes */
      seg$prefix$flag byte,
      lock$flag byte,
      rep$flag byte;

9  1  declare
      mod$bits byte,
      reg$bits byte,
      rm$bits byte,
      op$type byte,
      mnemonic$index byte,
      instr$1 byte,
      disp$len byte,        /* number of bytes in optional disp field */
      disp$word address;    /* value of optional disp field */

10 1  declare
      delimiters ($) byte data ('', ':', '+', cr, ' ', tab),
      left$bracket byte data ('['),
      right$bracket byte data (']'),
      num$delims literally '(length (delimiters) + 2)';

11 1  go$ddt: procedure external; /* return to assem loop in ddt on error */
12 2  end go$ddt;

13 1  ddt$set: procedure (assem$off, b) external;
14 2  declare assem$off address, b byte;

```

```

15 2      end ddt$set;

16 1      ddt$getline: procedure external;
17 2      end ddt$getline;

18 1      conin: procedure byte external;
19 2      end conin;

20 1      conout: procedure (b) external;
21 2      declare b byte;
22 2      end conout;

23 1      printa: procedure (a) EXTERNAL;
24 2      declare a address;
25 2      end printa;

26 1      crlf: procedure;
27 2      call conout (cr);
28 2      call conout (lf);
29 2      end crlf;

30 1      getline: procedure;
31 2      call ddt$getline;
32 2      next$ch = 0;
33 2      end getline;

34 1      print$word: procedure (a) EXTERNAL;
35 2      declare a address;
36 2      end print$word;

37 1      error: procedure;
38 2      call crlf;
39 2      call conout ('?');
40 2      call go$ddt;
41 2      end error;

42 1      ambig: procedure;
43 2      call printa (.(cr,lf,'ambiguous operand'));
44 2      call go$ddt;
45 2      end ambig;

46 1      check$ambig: procedure;
47 2      if (op1.w$bit and op2.w$bit) = 0ffh then call ambig;
48 2      if (op1.w$bit xor op2.w$bit) = 1 then call error;
49 2      end check$ambig;

52 1      compare: procedure (a, b, c) logical;
      /* should be able to replace this with a PL/M builtin */
53 2      declare (a, b) address, c byte;
54 2      declare a1 based a byte, b1 based b byte;
55 2      do while (c := c - 1) <> 255;
56 3          if a1 <> b1 then return false;
58 3          a = a + 1;
59 3          b = b + 1;
60 3      end;
61 2      return true;
62 2      end compare;

```

COPYRIGHT © 1981, 1982
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

```

/*****/
/* */
/* token scanning procedures */
/* */
/*****/

63 1  delimiter: procedure (b) logical;
64 2      declare b byte;
65 2      declare i byte;
66 2      do i = 0 to num$delims - 1;
67 3          if delimiters (i) = b then return true;
68 3      end;
69 2      return false;
70 2      end delimiter;

71 2

72 1  number: procedure (b) logical;
73 2      declare b byte;
74 2      return (b - '0' <= 9) or (b - 'A' <= 5);
75 2      end number;

76 1  hex: procedure (b) byte;
77 2      declare b byte;
78 2      if b - '0' <= 9 then return b - '0';
79 2      else return b - 'A' + 10;
80 2      end hex;

81 2

82 1  get$token: procedure;
83 2      declare numeric logical;
84 2      token$value = 0;
85 2      numeric = true;
86 2      token$len = 0;
87 2      if next$ch = 0 then next$ch = conin;
88 2      do while true;
89 3          token (token$len) = next$ch;
90 3          if (token$len := token$len + 1) >= length (token) then
91 4              call error;
92 3          if delimiter (next$ch) then
93 4              do;
94 5                  if token$len = 1 then
95 6                      do;
96 7                          token$type = delim$type;
97 7                          next$ch = 0;
98 7                      end;
99 5                  else
100 6                      do;
101 7                          token$len = token$len - 1;
102 7                          if numeric then token$type = number$type;
103 7                          else token$type = string$type;
104 7                      end;
105 6                      return;
106 6                  end;
107 5              if numeric then
108 6                  do;
109 7                      if number (next$ch) then token$value =
110 8                          shl (token$value,4) or hex (next$ch);
111 7                      else numeric = false;
112 7                  end;

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

113 4      end;
114 3      next$ch = conin;
115 3      end;
116 2      end get$token;

117 1      get$real$token: procedure;
118 2          do while true;
119 3              call get$token;
120 3              if token (0) > ' ' and token (0) <> tab then return;
122 3          end;
123 2      end get$real$token;

```

```

/*****
/*                                     */
/* operand scanning procedures      */
/*                                     */
*****/

```

```

124 1      look$up$reg: procedure (reg$tab$ptr, tab$len, reg$num$ptr) logical;
125 2          declare (reg$tab$ptr, reg$num$ptr) address;
126 2          declare reg$word based reg$tab$ptr address;
127 2          declare reg$num based reg$num$ptr byte;
128 2          declare tab$len byte;
129 2          declare i byte;
130 2          do i = 0 to tab$len - 1;
131 3              if token$word = reg$word then
132 3                  do;
133 4                      reg$num = i;
134 4                      return true;
135 4                  end;
136 3              reg$tab$ptr = reg$tab$ptr + 2;
137 3          end;
138 2          return false;
139 2      end look$up$reg;

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 570
 PACIFIC GROVE, CA 93950

SER. # _____

```

140 1      check$base$reg: procedure logical;
141 2          return look$up$reg (.base$reg, 2, .op.base$reg$num);
142 2      end check$base$reg;

143 1      check$index$reg: procedure logical;
144 2          return look$up$reg (.index$reg, 2, .op.index$reg$num);
145 2      end check$index$reg;

146 1      check$seg$reg: procedure logical;
147 2          if look$up$reg (.seg$reg, 4, .op.seg$reg$num) then
148 2              do;
149 3                  op.seg$reg$type = true;
150 3                  return true;
151 3              end;
152 2          return false;
153 2      end check$seg$reg;

154 1      check$reg$8: procedure logical;
155 2          if look$up$reg (.reg$8, 8, .op.reg$num) then
156 2              do;
157 3                  op.reg$8$type, op.addr$type = true;
158 3                  if op.reg$num = 0 then op.acc$type = true;

```

```

160 3      op.sub$type = 1;
161 3      op.w$bit = 0;
162 3      return true;
163 3      end;
164 2      return false;
165 2      end check$reg$8;

166 1      check$reg$16: procedure logical;
167 2      if look$up$reg (.reg$16, 8, .op.reg$ 16) then
168 2          do;
169 3              op.reg$16$type, op.modr$type = true;
170 3              if op.reg$num = 0 then op.acc$type = true;
172 3              op.sub$type = 2;
173 3              op.w$bit = 1;
174 3              return true;
175 3          end;
176 2      return false;
177 2      end check$reg$16;

178 1      check$b$prefix: procedure logical;
179 2      if token$type = string$type then
180 2          do;
181 3              if token$word = 'YB' then op.bw$prefix = 0;
183 3              else if token$word = 'OW' then op.bw$prefix = 1;
                  if op.bw$prefix <> 0ffh then return true;
187 3          end;
188 2      return false;
189 2      end check$b$prefix;

190 1      check$seg$prefix: procedure logical;
191 2      return check$seg$reg and (next$ch = ':');
192 2      end check$seg$prefix;

193 1      set$prefix: procedure (b);
194 2      declare b byte;
195 2      prefix (nprefix) = b;
196 2      nprefix = nprefix + 1;
197 2      end set$prefix;

198 1      set$seg$prefix: procedure;
199 2      if seg$prefix$flag then call error;
201 2      call set$prefix (26h or shl (op.seg$reg$num, 3));
202 2      seg$prefix$flag = true;
203 2      call get$token; /* eat ':' */
204 2      op.seg$reg$type = false; /* operand is not a seg reg */
205 2      end set$seg$prefix;

206 1      get$prefix: procedure;
207 2      do while true;
208 3          call get$real$token;
209 3          if token$type = delim$type then
210 3              do;
211 4                  if token (0) = cr then op.null$type = true;
213 4                  return;
214 4              end;
215 3          if check$seg$prefix then call set$seg$prefix;
217 3          else if check$b$prefix then

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

218 3      do;
219 4          if op.w$bit <> 0ffh then call error;
221 4          else op.w$bit = op.bw$prefix;
222 4      end;
223 3      else return; /* must not be a prefix */
224 3  end;
225 2  end get$prefix;

226 1  reg$indirect: procedure (x);
227 2      declare x byte;
228 2      op.modr$type = true;
229 2      if check$base$reg then
230 2          do;
231 3              call get$token;
232 3              if token (0) = right$bracket then
233 3                  do;
234 4                      op.modr$type = true;
235 4                      op.sub$type = 7 - x;
236 4                      return;
237 4                  end;
238 3              else if token (0) = '+' then
239 3                  do;
240 4                      op.sub$type = 9 - x;
241 4                      call get$token;
242 4                  end;
243 3              else call error;
244 3          end;
245 2      else op.sub$type = 8 - x;
246 2      if check$index$reg then
247 2          do;
248 3              call get$token;
249 3              if token (0) = right$bracket then return;
251 3          end;
252 2      call error;
253 2  end reg$indirect;

254 1  get$operand: procedure;
255 2      op.seg$reg$type, op.reg$16$type, op.reg$8$type, op.num$type,
          op.acc$type, op.null$type, op.for$type, op.modr$type = false;
256 2      op.bw$prefix, op.w$bit = 0ffh; /* indeterminate */
257 2      call get$prefix;
258 2      if op.null$type then return;
260 2      if check$seg$reg then return;
262 2      else if check$reg$8 then return;
264 2      else if check$reg$16 then return;
266 2      else if token$type = number$type then
267 2          do;
268 3          op.value = token$value;
269 3          if next$ch = left$bracket then
270 3              do;
271 4                  call get$token; /* eat '[' */
272 4                  call get$token;
273 4                  call reg$indirect (0);
274 4              end;
275 3          else if next$ch = ':' then
276 3              do;
277 4                  op.seg$value = token$value;

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____


```

278 4      call get$token; /* eat ':' */
279 4      call get$token;
280 4      if token$type <> number$type then call error;
282 4      op.value = token$value;
283 4      op.for$type = true;
284 4      end;
285 3      else op.num$type = true;
286 3      end;
      else
287 2      do;
288 3      if token (0) <> left$bracket then call error;
290 3      call get$token;
291 3      if token$type = number$type then
292 3      do;
293 4      op.value = token$value;
294 4      call get$token;
295 4      if token (0) = right$bracket then
296 4      do;
297 5      op.sub$type = 3;
298 5      op.modr$type = true;
299 5      end;
300 4      else call error;
301 4      end;
302 3      else call reg$indirect (3);
303 3      end;
304 2      end get$operand;

```

```

305 1      get$operand$1: procedure;
306 2      op$base = .op1;
307 2      call get$operand;
308 2      call get$token;
309 2      if token (0) <> ',' then call error;
311 2      end get$operand$1;

```

```

312 1      get$operand$2: procedure;
313 2      op$base = .op2;
314 2      call get$operand;
315 2      if op.null$type then return;
317 2      call get$token;
318 2      if token (0) <> cr then call error;
320 2      end get$operand$2;

```

```

/*****
/*                                     */
/* opcode scanning procedures         */
/*                                     */
*****/

```

```

321 1      lookup$optab: procedure;
322 2      declare i address;
323 2      do i = 0 to last (optab) by 4;
324 3      if optab (i) = anemonic$index then
325 3      do;
326 4      op$type = op$tab (i+1);
327 4      instr$1 = op$tab (i+2);
328 4      reg$bits = op$tab (i+3);
329 4      return;

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

330 4         end;
331 3     end;
332 2     call error; /* should never get here */
333 2     end lookup$opstab;

334 1 valid$mnemonic: procedure logical;
335 2     declare
        i byte,
        table$base address,
        mnemonic based table$base byte;
336 2     if token$len - 2 > 4 then call error;
338 2     table$base = tab$ptrs (token$len - 2);
339 2     do i = 0 to nops (token$len - 2) - 1;
340 3         if compare (.token, .mnemonic, token$len) then
341 3             do;
342 4                 mnemonic$index = i + opn$in (token$len - 2);
343 4                 return true;
344 4             end;
345 3         table$base = table$base + token$len;
346 3     end;
347 2     return false;
348 2     end valid$mnemonic;

349 1 get$opcode: procedure logical;
350 2     nprefix = 0;
351 2     rep$flag, lock$flag, seg$prefix$flag = false;
352 2     do while true;
353 3         call get$real$token;
354 3         if token (0) = cr or token (0) = '.' then return false; /* no opcode */
356 3         if check$seg$prefix then call set$seg$prefix;
358 3         else if valid$mnemonic then
359 3             do;
360 4                 call lookup$opstab;
361 4                 if op$type = 0ffh then
362 4                     do;
363 5                         if lock$flag then call error;
365 5                         lock$flag = true;
366 5                         call set$prefix (instr$1);
367 5                     end;
368 4                 else if op$type = 0feh then /* repeat */
369 4                     do;
370 5                         if rep$flag then call error;
372 5                         rep$flag = true;
373 5                         call set$prefix (instr$1);
374 5                     end;
375 4                 else return true; /* opcode present */
376 4             end;
377 3         else call error;
378 3     end;
379 2     end get$opcode;

```

```

/*****
/*
/* forming and output of instructions */
/*
*****/

```

COPYRIGHT © 1981, 1982
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

```

380 1      put$mem: procedure (b);
381 2          declare b byte;
382 2          call ddt$set (assem$offset, b);
383 2          assem$offset = assem$offset + 1;
384 2          end put$mem;

385 1      put$prefix: procedure;
386 2          declare i byte;
387 2          i = 0;
388 2          do while (nprefix := nprefix - 1) <> 0ffh;
389 3              call put$mem (prefix (i));
390 3              i = i + 1;
391 3          end;
392 2          end put$prefix;

393 1      put$instr: procedure;
394 2          call put$prefix;
395 2          call put$mem (instr$1);
396 2          end put$instr;

397 1      set$instr: procedure (b);
398 2          declare b byte;
399 2          instr$1 = b;
400 2          call put$instr;
401 2          end set$instr;

402 1      put$word: procedure (a);
403 2          declare a address;
404 2          call put$mem (low (a));
405 2          call put$mem (high (a));
406 2          end put$word;

407 1      set$w$bit: procedure;
408 2          if (op1.w$bit and op2.w$bit) = 1 then instr$1 = instr$1 or 1;
410 2          end set$w$bit;

411 1      set$mod: procedure;
412 2          if op.value > 7FH then mod$bits, disp$len = 2;
414 2          else mod$bits, disp$len = 1;
415 2          end set$mod;

416 1      set$modrm: procedure;
417 2          disp$len = 0;
418 2          disp$word = op.value;
419 2          do case op.sub$type - 1;
420 3              do; /* 1 */
421 4                  mod$bits = 3;
422 4                  rm$bits = op.reg$num;
423 4              end;
424 3              do; /* 2 */
425 4                  mod$bits = 3;
426 4                  rm$bits = op.reg$num;
427 4              end;
428 3              do; /* 3 */
429 4                  mod$bits = 0;
430 4                  rm$bits = 6;
431 4                  disp$len = 2;

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

432 4      end;
433 3      do; /* 4 */
434 4          if op.base$reg$num = 1 then /* BP */
435 4              do;
436 5                  mod$bits = 1;
437 5                  rm$bits = 6;
438 5                  disp$len = 1;
439 5                  disp$word = 0;
440 4              end;
441 4          else
442 5              do;
443 5                  mod$bits = 0;
444 5                  rm$bits = 7;
445 4              end;
446 3      do; /* 5 */
447 4          mod$bits = 0;
448 4          rm$bits = op.index$reg$num + 4;
449 4      end;
450 3      do; /* 6 */
451 4          mod$bits = 0;
452 4          rm$bits = op.base$reg$num * 2 + op.index$reg$num;
453 4      end;
454 3      do; /* 7 */
455 4          call set$mod;
456 4          rm$bits = 7 - op.base$reg$num;
457 4      end;
458 3      do; /* 8 */
459 4          call set$mod;
460 4          rm$bits = op.index$reg$num + 4;
461 4      end;
462 3      do; /* 9 */
463 4          call set$mod;
464 4          rm$bits = op.base$reg$num * 2 + op.index$reg$num;
465 4      end;
466 3      end;
467 2      end set$modrm;

468 1      do$modrm: procedure (op$ptr);
469 2          declare op$ptr address;
470 2          op$base = op$ptr;
471 2          call set$modrm;
472 2          call put$instr;
473 2          call put$mem (shl (mod$bits, 6) or shl (reg$bits, 3) or rm$bits);
474 2          if disp$len > 0 then call put$mem (low (disp$word));
475 2          if disp$len > 1 then call put$mem (high (disp$word));
476 2          end do$modrm;

479 1      modrm$w: procedure (ins, reg, op$ptr);
480 2          declare (ins, reg) byte;
481 2          declare op$ptr address;
482 2          op$base = op$ptr;
483 2          if op.modrm$type then
484 2              do;
485 3                  if op.w$bit = 0ffh then call ambig;
486 3                  instr$1 = ins or op.w$bit;
487 3                  reg$bits = reg;

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

489 3      call do$modrm (op$ptr);
490 3      end;
491 2      else call error;
492 2      end modrm$w;

493 1      modrm$16: procedure (ins, reg);
494 2      declare (ins, reg) byte;
495 2      if op2.modrm$type and not op2.reg$8$type then
496 2      do;
497 3          instr$1 = ins;
498 3          reg$bits = reg;
499 3          call do$modrm (.op2);
500 3      end;
501 2      else call error;
502 2      end modrm$16;

503 1      modregm: procedure;
504 2      call check$ambig;
505 2      if op1.sub$type <= 2 then /* first op is register */
506 2      do;
507 3          reg$bits = op1.reg$num;
508 3          call do$modrm (.op2);
509 3      end;
510 2      else
511 3          instr$1 = instr$1 or op2.w$bit;
512 3          reg$bits = op2.reg$num;
513 3          call do$modrm (.op1);
514 3      end;
515 2      end modregm;

516 1      modregm$w: procedure;
517 2      call set$w$bit;
518 2      call modregm;
519 2      end modregm$w;

520 1      modregm$dw: procedure;
521 2      if op1.modrm$type and (op1.sub$type <= 2) then instr$1 = instr$1 or 2; /* set d
    bit */
523 2      call modregm$w;
524 2      end modregm$dw;

525 1      put$immed: procedure;
526 2      call put$mem (low (op2.value));
527 2      if op1.w$bit then call put$mem (high (op2.value));
529 2      end put$immed;

530 1      put$immed$s: procedure;
531 2      call put$mem (low (op2.value));
532 2      if (instr$1 and 3) = 1 then call put$mem (high (op2.value));
534 2      end put$immed$s;

535 1      check$w$and$val: procedure;
536 2      if op1.w$bit = 0 and op2.value > 255 then call error;
538 2      if op1.w$bit = 0ffh then
539 2      do;
540 3          if op2.value >= 256 then op1.w$bit = 1;

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

542 3      else call ambig;
543 3      end;
544 2      end check$w$and$val;

545 1      acc$immed: procedure;
546 2      call check$w$and$val;
547 2      call set$w$bit;
548 2      call put$instr;
549 2      call put$immed;
550 2      end acc$immed;

551 1      modr$immed: procedure;
552 2      call check$w$and$val;
553 2      call set$w$bit;
554 2      call do$modr (.op1);
555 2      call put$immed;
556 2      end modr$immed;

557 1      set$instr$w: procedure (b);
558 2      declare b byte;
559 2      if op2.sub$type <= 2 then instr$1 = b or op2.w$bit;
561 2      else instr$1 = b or op1.w$bit;
562 2      end set$instr$w;

563 1      check$acc$immed: procedure logical;
564 2      return op1.acc$type and op2.num$type;
565 2      end check$acc$immed;

566 1      check$modr$immed: procedure logical;
567 2      return op1.modr$type and op2.num$type;
568 2      end check$modr$immed;

569 1      check$modr$modr: procedure logical;
570 2      return op1.modr$type and op2.modr$type and not
        (op1.sub$type > 2 and op2.sub$type > 2);
571 2      end check$modr$modr;

/*****
/*
/* mnemonic processing procedures */
/*
*****/

572 1      type1: procedure;
        /*
        mnemonics: simple one-byte instructions
        operands: none
        forms:      xxxxxxxx
        */
573 2      call get$operand$2;
574 2      if op2.null$type then call put$instr;
576 2      else call error;
577 2      end type1;

578 1      type2: procedure;
        /*
        mnemonics: add aam

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

        operands: none
        forms:      xxxxxxxx 00001010
    */
579 2    call get$operand$2;
580 2    if op2.null$type then
581 2        do;
582 3        call put$instr;
583 3        call put$mem (0ah);
584 3        end;
585 2    else call error;
586 2    end type2;

587 1    type3: procedure;
    /*
        mnemonics: conditional jumps loopxx japs
        operands:  1 (number)
        forms:      xxxxxxxx IP-INCB
    */
588 2    call get$operand$2;
589 2    if op2.num$type and (op2.value - (assem$offset - 7eh) <= 255) then
590 2        do;
591 3        call put$instr;
592 3        call put$mem (low (op2.value - (assem$offset + 1)));
593 3        end;
594 2    else call error;
595 2    end type3;

596 1    type4: procedure;
    /*
        mnemonics: logical shifts and rotates
        operands:  2
        forms:      xxxxxxvw rrrrr/m [disp-lo] [disp-hi]
    */
597 2    call get$operand$1;
598 2    call get$operand$2;
599 2    if op1.adr$type then
600 2        do;
601 3        if op2.reg$type and (op2.reg$num = 1) then
602 3            instr$1 = instr$1 or 2; /* set v bit */
603 3        else if not (op2.num$type and (op2.value = 1)) then call error;
604 3        call modrm$w (instr$1, reg$bits, .op1);
605 3        end;
606 2    else call error;
607 2    end type4;

608 2    end type4;

609 1    type5: procedure;
    /*
        mnemonics: div idiv mul imul not neg
        operands:  1
        forms:      xxxxxxvw rrrrr/m [disp-lo] [disp-hi]
    */
610 2    call get$operand$2;
611 2    call modrm$w (instr$1, reg$bits, .op2);
612 2    end type5;

613 1    type6: procedure;
    /*

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

        mnemonics: les lds lea
        operands: 2 (reg16, memory)
        forms:      xxxxxxxx mregr/m [disp-lo] [disp-hi]
    */
614 2    call get$operand$1;
615 2    call get$operand$2;
616 2    if op1.reg$16$type and op2.modrm$type and (op2.sub$type > 2) then
617 2        call mod$reg$r;
618 2    else call error;
619 2    end type6;

620 1    declare
        type$7$struct (2) structure (op$val (2) byte) initial (
            048h, 001h, /* dec */
            040h, 000h); /* inc */

621 1    type7: procedure;
        /*
            mnemonics: inc dec
            operands: 1
            forms:      xxxxxreg
                      xxxxxxxw mxxxr/m [disp-lo] [disp-hi]
        */
622 2    call get$operand$2;
623 2    if op2.reg$16$type then call set$instr (type$7$struct (instr$1).op$val (0) or op2.r
eg$num);
625 2    else call modrm$w (0feh, type$7$struct (instr$1).op$val (1), op2);
626 2    end type7;

627 1    type8or8a: procedure;
628 2    if op2.seg$reg$type then call set$instr (type$8$struct (instr$1).op$val (0) or
shl (op2.seg$reg$num, 3));
630 2    else if op2.reg$16$type then call set$instr (type$8$struct (instr$1).op$val (1) or
op2.reg$num);
632 2    else call modrm$16 (type$8$struct (instr$1).op$val (2),
type$8$struct (instr$1).op$val (3));
633 2    end type8or8a;

634 1    declare
        type$8$struct (2) structure (op$val (4) byte) initial (
            07h, 58h, 8fh, 00h, /* pop */
            06h, 50h, 0fh, 06h); /* push */

635 1    type8: procedure;
        /*
            mnemonics: push
            operands: 1
            forms:      xxxSRxxx
                      xxxxxreg
                      xxxxxxxw mxxxr/m [disp-lo] [disp-hi]
        */
636 2    call get$operand$2;
637 2    call type8or8a;
638 2    end type8;

639 1    type8a: procedure;
        /*

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____


```

      mnemonics: pop
      operands: 1
      forms:      xxxSRxxx
                  xxxxxreg
                  xxxxxxxx mxxxr/m [disp-lo] [disp-hi]
*/
640 2      call get$operand2;
641 2      if op2.seg$reg$type and (op2.seg$reg$num = 1) then call error;
643 2      else call type8or8a;
644 2      end type3a;

645 1      type9or10: procedure (type);
646 2      declare type byte;
647 2      call get$operand1;
648 2      call get$operand2;
649 2      if check$acc$immed then
650 2          do;
651 3              instr$1 = instr$1 or 4;
652 3              call acc$immed;
653 3          end;
654 2      else if check$modr$immed then
655 2          do;
656 3              instr$1 = 80h;
657 3              if type = 9 then
658 3                  do;
659 4                      if op2.value < 80h or op2.value >= 0ff80h then
660 4                          instr$1 = instr$1 or 2;
661 4                          call check$w$and$val;
662 4                          call set$w$bit;
663 4                          call do$modrm (.op1);
664 4                          call put$immed$;
665 4                      end;
666 3                  else call modr$immed;
667 3              end;
668 2      else if check$modrm$modrm then call mod$reg$rm$dw;
670 2      else call error;
671 2      end type9or10;

672 1      type9: procedure;
/*
      mnemonics: add adc cmp sub sbb
      operands: 2
      forms:      xxxxxxxw data-lo data-hi (if w=1)
                  xxxxxxxw mxxxr/m [disp-lo] [disp-hi] data-lo data-hi (if sw=
- 01)
                  xxxxxxw mreg/r/m [disp-lo] [disp-hi]
*/
673 2      call type9or10 (9);
674 2      end type9;

675 1      type10: procedure;
/*
      mnemonics: and or xor
      operands: 2
      forms:      xxxxxxxw data-lo data-hi (if w=1)
                  xxxxxxxw mxxxr/m [disp-lo] [disp-hi] data-lo data-hi (if w=1
- )

```

CP/M-86 v.1.1 (vol. III)

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # C81-162

```

                                xxxxxdw amregr/m [disp-lo] [disp-hi]
                                */
676 2    call type9or10 (10);
677 2    end type10;

678 1    declare type11$struct (2) structure (op$val (2) byte) initial (
                                0e8h, 2, /* call */
                                0e7h, 4); /* jmp */

679 1    type11: procedure;
                                /*
                                mnemonics: call jmp
                                operands: 1
                                forms:      xxxxxxxx IP-INC16
                                           xxxxxxxx amxxxr/m [disp-lo] [disp-hi]
                                */
680 2    call get$operand$2;
681 2    if op2.num$type then
682 2        do;
683 3        call set$instr (type11$struct (instr$1).op$val (0));
684 3        call put$word (op2.value - (assem$offset + 2));
685 3        end;
686 2    else call modrm$16 (0ffh, type11$struct (instr$1).op$val (1));
687 2    end type11;

688 1    type$int: procedure;
689 2    call get$operand$2;
690 2    if op2.num$type and (op2.value <= 255) then
691 2        do;
692 3        if op2.value = 3 then call set$instr (0cch);
693 3        else
694 3            do;
695 4            call set$instr (0cdh);
696 4            call put$mem (low (op2.value));
697 4            end;
698 3        end;
699 2    else call error;
700 2    end type$int;

701 1    type$esc: procedure;
702 2    call get$operand$1;
703 2    call get$operand$2;
704 2    if op1.num$type and (op1.value < 64) and op2.modrm$type then
705 2        do;
706 3        instr$1 = instr$1 or shr (low (op1.value), 3);
707 3        reg$bits = low (op1.value) and 7;
708 3        call do$modrm (.op2);
709 3        end;
710 2    else call error;
711 2    end type$esc;

712 1    type$ret$retf: procedure;
713 2    op$base = .op2;
714 2    call get$operand;
715 2    if op2.null$type then call set$instr (instr$1 or 1);
717 2    else if op2.num$type then
718 2        do;

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

719 3      call put$instr;
720 3      call put$word (op2.value);
721 3      end;
722 2      else call error;
723 2      end type$ret$retf;

724 1      type$in: procedure;
725 2          call get$operand1;
726 2          if not op1.acc$type then call error;
728 2          call get$operand2;
729 2          if op2.reg$16$type and (op2.reg$num = 2) /* DX */ then
730 2              call set$instr (instr$1 or 8 or op1.w$bit);
731 2          else if op2.num$type and (op2.value <= 255) then
732 2              do;
733 3                  call put$instr;
734 3                  call put$mem (low (op2.value));
735 3              end;
736 2          else call error;
737 2          end type$in;

738 1      type$out: procedure;
739 2          declare temp byte;
740 2          call get$operand1;
741 2          if op1.reg$16$type and (op1.reg$num = 2) /* DX */ then temp = 1;
743 2          else if op1.num$type and (op1.value <= 255) then temp = 0;
745 2          else call error;
746 2          call get$operand2;
747 2          if not op2.acc$type then call error;
749 2          call put$mem (instr$1 or shl (temp,3) or op2.w$bit);
750 2          if temp = 0 then call put$mem (low (op1.value));
752 2          end type$out;

753 1      declare cjstruc (2) structure (op$val (2) byte) initial (
          09ah, 3, /* callf */
          0eah, 5); /* jmpf */

754 1      type$callf$jmpf: procedure;
755 2          call get$operand2;
756 2          if op2.far$type then
757 2              do;
758 3                  call set$instr (cjstruc (instr$1).op$val (0));
759 3                  call put$word (op2.value);
760 3                  call put$word (op2.seg$value);
761 3              end;
762 2          else call modrm$16 (0ffh, cjstruc (instr$1).op$val (1));
763 2          end type$callf$jmpf;

764 1      type$test: procedure;
765 2          call get$operand1;
766 2          call get$operand2;
767 2          if check$acc$immed then
768 2              do;
769 3                  instr$1 = 0a8h;
770 3                  call acc$immed;
771 3              end;
772 2          else if check$modrm$immed then
773 2              do;

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

774 3      instr$1 = 0f6h;
775 3      call modrm$immed;
776 3      end;
777 2      else if check$modrm$modrm then
778 2          do;
779 3          call set$instr$w (84h);
780 3          call mod$reg$rm;
781 3          end;
782 2      else call error;
783 2      end type$test;

784 1      type$xchg; procedure;
785 2          call get$operand$1;
786 2          call get$operand$2;
787 2          if op1.acc$type and op1.w$bit and op2.reg$16$type then
788 2              call set$instr (90h + op2.reg$num);
789 2          else if check$modrm$modrm then
790 2              do;
791 3              call set$instr$w (86h);
792 3              call modreg$rm;
793 3              end;
794 2          else call error;
795 2          end type$xchg;

796 1      type$mov; procedure;
797 2          call get$operand$1;
798 2          call get$operand$2;
799 2          if op1.seg$reg$type and op2.modrm$type and not op2.reg$8$type then
800 2              do;
801 3              instr$1 = 8eh;
802 3              reg$bits = op1.seg$reg$num;
803 3              call do$modrm (.op2);
804 3              end;
805 2          else if op1.modrm$type and not (op1.sub$type = 1) and op2.seg$reg$type then
806 2              do;
807 3              instr$1 = 8ch;
808 3              reg$bits = op2.seg$reg$num;
809 3              call do$modrm (.op1);
810 3              end;
811 2          else if op1.acc$type and op2.modrm$type and (op2.sub$type = 3) then
812 2              do;
813 3              instr$1 = 0a0h or op1.w$bit;
814 3              call put$instr;
815 3              call put$word (op2.value);
816 3              end;
817 2          else if op2.acc$type and op1.modrm$type and (op1.sub$type = 3) then
818 2              do;
819 3              instr$1 = 0a2h or op2.w$bit;
820 3              call put$instr;
821 3              call put$word (op1.value);
822 3              end;
823 2          else if check$modrm$immed then
824 2              do;
825 3              if op1.sub$type <= 2 then
826 3                  do;
827 4                  call set$instr (0b0h or shl (op1.w$bit, 3) or op1.reg$num);
828 4                  call put$immed;

```

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

829 4          end;
          else
830 3          do;
831 4              instr$1 = 0c6h;
832 4              call modra$inmed;
833 4          end;
834 3      end;
835 2      else if check$modra$modrm then
836 2      do;
837 3          instr$1 = 88h;
838 3          call mod$reg$ra$dw;
839 3      end;
840 2      else call error;
841 2      end type$mov;

842 1  asm: procedure (loc) address public;
843 2      declare loc pointer;
844 2      assem$ptr = loc;
845 2      call crlf;
846 2      call print$word (assem$segment);
847 2      call conout (' ');
848 2      call print$word (assem$offset);
849 2      call conout (' ');
850 2      call get$line;
851 2      op$base = .opi; /* must be set for prefix scan */
852 2      if get$opcode then
853 2          do;
854 3          if op$type > 21 then call error;
855 3          do case op$type;
856 3              call error;
857 4              call type1;
858 4              call type2;
859 4              call type3;
860 4              call type4;
861 4              call type5;
862 4              call type6;
863 4              call type7;
864 4              call type8;
865 4              call type9;
866 4              call type10;
867 4              call type11;
868 4              call type$int;
869 4              call type$esc;
870 4              call type$ret$retf;
871 4              call type$in;
872 4              call type$out;
873 4              call type$callf$jmpf;
874 4              call type$test;
875 4              call type$xchg;
876 4              call type$mov;
877 4              call type$mov;
878 4              call type$mov;
879 4          end;
880 3          end;
881 2      else call put$prefix; /* may be prefix without opcode */
882 2      return assem$offset;
883 2      end asm;

```

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

884 1 end ddtasm;

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

CROSS-REFERENCE LISTING

DEFN	ADDR	SIZE	NAME, ATTRIBUTES, AND REFERENCES
------	------	------	----------------------------------

34	0000H	2	A. WORD PARAMETER 35
52	0008H	2	A. WORD PARAMETER AUTOMATIC
23	0000H	2	A. WORD PARAMETER 24
402	0004H	2	A. WORD PARAMETER AUTOMATIC
54	0000H	1	A1 BYTE BASED(A) 56
545	09CAH	17	ACCINMED PROCEDURE STACK=0016H 652 770
7	0008H	1	ACCTYPE. BYTE MEMBER(OP) 159 171 255
7	0008H	1	ACCTYPE. BYTE MEMBER(OP2) 747 817
7	0008H	1	ACCTYPE. BYTE MEMBER(OP1) 564 726 787 811
42	0031H	15	AMBIG. PROCEDURE STACK=0006H 48 486 542
842	0FC8H	248	ASM. PROCEDURE WORD PUBLIC STACK=0036H
5	0000H	1	ASSEMBYTE. BYTE BASED(ASSEMBPTR)
13	0000H	2	ASSEMBOFF WORD PARAMETER 14
5	0002H	2	ASSEMBOFFSET. WORD AT 382 383 589 592 684 848 882
5	0002H	4	ASSEMBPTR POINTER 5 844
5	0004H	2	ASSEMBSEGMENT WORD AT 846
72	0004H	1	B. BYTE PARAMETER AUTOMATIC 73 74
52	0006H	2	B. WORD PARAMETER AUTOMATIC 53 54 56 59
13	0000H	1	B. BYTE PARAMETER 14
557	0004H	1	B. BYTE PARAMETER AUTOMATIC 558 560 561
63	0004H	1	B. BYTE PARAMETER AUTOMATIC 64 67
397	0004H	1	B. BYTE PARAMETER AUTOMATIC 398 399
20	0000H	1	B. BYTE PARAMETER 21
380	0004H	1	B. BYTE PARAMETER AUTOMATIC 381 382
76	0004H	1	B. BYTE PARAMETER AUTOMATIC 77 78 79 80
193	0004H	1	B. BYTE PARAMETER AUTOMATIC 194 195
54	0000H	1	B1 BYTE BASED(B) 56
6	0000H	4	BASEREG. BYTE ARRAY(4) DATA 141
7	0002H	1	BASEREGNUM BYTE MEMBER(OP2)
7	0002H	1	BASEREGNUM BYTE MEMBER(OP1)
7	0002H	1	BASEREGNUM BYTE MEMBER(OP) 141 434 452 456 464
7	000DH	1	BWPREFIX BYTE MEMBER(OP2)
7	000DH	1	BWPREFIX BYTE MEMBER(OP1)
7	000DH	1	BWPREFIX BYTE MEMBER(OP) 182 184 185 221 256
52	0004H	1	C. BYTE PARAMETER AUTOMATIC 53 55
563	0A11H	12	CHECKACCINMED. PROCEDURE BYTE STACK=0002H 649 767
46	0040H	33	CHECKAMBIG PROCEDURE STACK=000EH 504
140	01F1H	23	CHECKBASEREG PROCEDURE BYTE STACK=000CH 229
178	02E3H	62	CHECKBWPREFIX. PROCEDURE BYTE STACK=0002H 217
143	0208H	23	CHECKINDEXREG. PROCEDURE BYTE STACK=000CH 246
566	0A1DH	12	CHECKMODRNIMMED. PROCEDURE BYTE STACK=0002H 654 772 823
569	0A29H	42	CHECKMODRNMODRM. PROCEDURE BYTE STACK=0006H 668 777 789
166	0295H	78	CHECKREG16 PROCEDURE BYTE STACK=000CH 264
154	0247H	78	CHECKREG8. PROCEDURE BYTE STACK=000CH 262
190	0321H	22	CHECKSEGPREFIX PROCEDURE BYTE STACK=0010H 215 356
146	021FH	40	CHECKSEGREG. PROCEDURE BYTE STACK=000CH 191 260

COPYRIGHT 1981, 1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

53 54 56 58
SER. =

535	099AH	48	CHECKWANDVAL . . .	PROCEDURE STACK=000EH	546	552	661
753	001EH	4	CJSTRUC.	STRUCTURE ARRAY(2) INITIAL	758	762	
52	0061H	45	COMPARE.	PROCEDURE BYTE STACK=0008H	340		
18	0000H		CONIN.	PROCEDURE BYTE EXTERNAL(10) STACK=0000H		88	
				114			
20	0000H		CONOUT	PROCEDURE EXTERNAL(11) STACK=0000H	27	28	
				39 847 849			
2			CR	LITERALLY	10	27	43 211 318 354
26	0002H	17	CRLF	PROCEDURE STACK=0006H	38	845	
1	0002H		DDTASM	PROCEDURE STACK=0000H			
16	0000H		DDTGETLINE	PROCEDURE EXTERNAL(9) STACK=0000H		31	
13	0000H		DDTSET	PROCEDURE EXTERNAL(8) STACK=0000H		382	
63	008EH	46	DELIMITER.	PROCEDURE BYTE STACK=0004H	93		
10	0008H	6	DELIMITERS	BYTE ARRAY(6) DATA	66	67	
4			DELINTYPE.	LITERALLY	97	209	
9	0078H	1	DISPLEN.	BYTE	413	414	417 431 438 474 476
9	0008H	2	DISPWORD	WORD	418	439	475 477
468	0859H	74	DOMODRM.	PROCEDURE STACK=0018H	489	499	508 513
				554 663 708 803 809			
37	0020H	17	ERROR.	PROCEDURE STACK=000AH	50	92	200 220
				243 252 281 289 300 310 319 332 337 364			
				371 377 491 501 537 576 585 594 604 607			
				618 642 670 699 710 722 727 736 745 748			
				782 794 840 855 857			
2			FALSE.	LITERALLY	57	70	112 138 152 164 176
				188 204 255 347 351 355			
7	000AH	1	FARTYPE.	BYTE MEMBER(OP1)			
7	000AH	1	FARTYPE.	BYTE MEMBER(OP2)	756		
7	000AH	1	FARTYPE.	BYTE MEMBER(OP)	255	283	
30	0013H	13	GETLINE.	PROCEDURE STACK=0004H	850		
349	066EH	126	GETOPCODE.	PROCEDURE BYTE STACK=0016H	852		
254	0445H	295	GETOPERAND	PROCEDURE STACK=001AH	307	314	714
305	056CH	27	GETOPERAND1. . . .	PROCEDURE STACK=001EH	597	614	647 702
				725 740 765 785 797			
312	0587H	40	GETOPERAND2. . . .	PROCEDURE STACK=001EH	573	579	588 598
				610 615 622 636 640 648 680 689 703 728			
				746 755 766 786 798			
206	037EH	80	GETPREFIX.	PROCEDURE STACK=0016H	257		
117	01A0H	24	GETREALTOKEN . . .	PROCEDURE STACK=0012H	208	353	
82	00F8H	168	GETTOKEN	PROCEDURE STACK=000EH	119	203	231 241
				248 271 272 278 279 290 294	308	312	
11	0000H		GODDT.	PROCEDURE EXTERNAL(7) STACK=0000H		40	44
76	00E1H	23	HEX.	PROCEDURE BYTE STACK=0004H	111		
			HIGH	BUILTIN	405	477	528 533
335	007CH	1	I.	BYTE	339	342	
129	007BH	1	I.	BYTE	130	133	
65	0077H	1	I.	BYTE	66	67	
322	000AH	2	I.	WORD	323	324	326 327 328
386	007DH	1	I.	BYTE	387	389	390
6	0004H	4	INDEXREG	BYTE ARRAY(4) DATA	144		
7	0003H	1	INDEXREGNUM. . . .	BYTE MEMBER(OP2)			
7	0003H	1	INDEXREGNUM. . . .	BYTE MEMBER(OP1)			
7	0003H	1	INDEXREGNUM. . . .	BYTE MEMBER(OP)	144	448	452 460 464
479	0008H	1	INS.	BYTE PARAMETER AUTOMATIC	480	487	
493	0003H	1	INS.	BYTE PARAMETER AUTOMATIC	494	497	
9	0077H	1	INSTR1	BYTE	327	366	373 395 399 409 487 497
				511 522 532 560 561 602 605 611 624 625			

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P.O. BOX 570
 PACIFIC GROVE, CA 93950

SEE PAGE

			629 631 632 651 656 660 683 686 706 716
			730 749 758 762 769 774 801 807 813 819
			831 837
		LAST	BUILTIN 323
10 000EH	1	LEFTBRACKET. . . .	BYTE DATA 269 288
		LENGTH	BUILTIN 66 91
2		LF	LITERALLY 28 43
842 0004H	4	LOC.	POINTER PARAMETER AUTOMATIC 843 844
8 0070H	1	LOCKFLAG	BYTE 351 363 365
2		LOGICAL.	LITERALLY 7 52 63 72 83 124 140
			143 146 154 166 178 190 334 349 563 566
			569
321 05AFH	68	LOOKUPOPTAB. . . .	PROCEDURE STACK=000EH 360
124 01B8H	57	LOOKUPREG.	PROCEDURE BYTE STACK=000BH 141 144 147
			155 167
		LOW.	BUILTIN 404 475 526 531 592 696 706
			707 734 751
335 0000H	1	MNEMONIC	BYTE BASED(TABLEBASE) 340
9 0076H	1	MNEMONICINDEX. . .	BYTE 324 342
9 0072H	1	MODBITS.	BYTE 413 414 421 425 429 436 442 447
			451 473
503 090CH	46	MODREGRM	PROCEDURE STACK=001CH 518 617 780 792
520 094SH	27	MODREGRMOW	PROCEDURE STACK=0024H 669 838
516 093AH	11	MODREGRMW.	PROCEDURE STACK=0020H 523
493 08E0H	44	MODRM16.	PROCEDURE STACK=0020H 632 686 762
551 09DBH	21	MODRMIMMED	PROCEDURE STACK=001CH 666 775 832
7 000BH	1	MODRMTYPE.	BYTE MEMBER(OP2) 495 570 616 704 799
			811
7 000BH	1	MODRMTYPE.	BYTE MEMBER(OP1) 521 567 570 599 805
			817
7 000BH	1	MODRMTYPE.	BYTE MEMBER(OP) 157 169 228 234 255 298
			483
479 08A3H	61	MODRMW	PROCEDURE STACK=0022H 605 611 625
4 0044H	1	NEXTCH	BYTE 32 87 88 90 93 98 110 111
			114 191 269 275
3 0000H	5	NOPS	BYTE ARRAY(5) EXTERNAL(1) 339
3 006EH	1	NPREFIX.	BYTE 195 196 350 388
7 0009H	1	NULLTYPE	BYTE MEMBER(OP1)
7 0007H	1	NULLTYPE	BYTE MEMBER(OP2) 574 580 715
7 0009H	1	NULLTYPE	BYTE MEMBER(OP) 212 255 258 315
72 00BCH	37	NUMBER	PROCEDURE BYTE STACK=0006H 110
4		NUMBERTYPE	LITERALLY 103 266 280 291
10		NUMDELIMS.	LITERALLY 66
83 007AH	1	NUMERIC.	BYTE 85 102 108 112
7 0007H	1	NUMTYPE.	BYTE MEMBER(OP2) 564 567 589 603 681
			690 717 731
7 0007H	1	NUMTYPE.	BYTE MEMBER(OP1) 704 743
7 0007H	1	NUMTYPE.	BYTE MEMBER(OP) 255 285
7 0000H	19	OP	STRUCTURE BASED(OPBASE) 141 144 147 149
			155 157 158 159 160 161 167 169 170 171
			172 173 182 184 185 201 204 212 219 221
			228 234 235 240 245 255 256 258 268 277
			282 283 285 293 297 298 315 412 418 419
			422 426 434 448 452 456 460 464 483 485
			487
7 0045H	19	OP1.	STRUCTURE 47 49 306 408 505 507 513
			521 527 536 538 541 554 561 564 567 570

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P.O. BOX 579

SUNSHINE GROVE, CA 95950

				599	605	616	663	704	706	707	726	730	741
				743	751	787	799	802	805	809	811	813	817
				821	825	827	851						
7	0058H	19	OP2.	STRUCTURE		47	49	313	408	495	499	508	
				511	512	526	528	531	533	536	540	559	560
				564	567	570	574	580	589	592	601	603	611
				616	623	624	625	628	629	630	631	641	659
				681	684	690	692	696	704	708	713	715	717
				720	729	731	734	747	749	756	759	760	787
				788	799	803	805	808	811	815	817	819	
7	0006H	2	OPBASE.	WORD		7	141	144	147	149	155	157	158
				159	160	161	167	169	170	171	172	173	182
				184	185	201	204	212	219	221	228	234	235
				240	245	255	256	258	268	277	282	283	285
				293	297	298	306	313	315	412	418	419	422
				426	434	448	452	456	460	464	470	482	483
				485	487	713	851						
3	0000H	6	OPWIN.	BYTE ARRAY(6) EXTERNAL(2)					342				
479	0004H	2	OPPTR.	WORD PARAMETER AUTOMATIC					481	482	489		
468	0004H	2	OPPTR.	WORD PARAMETER AUTOMATIC					469	470			
3	0000H	480	OPTAB.	BYTE ARRAY(480) EXTERNAL(3)						323	324	326	
				327	328								
9	0075H	1	OPTYPE.	BYTE		326	361	368	854	856			
753	0000H	2	OPVAL.	BYTE ARRAY(2) MEMBER(CJSTRUC)						758	762		
634	0000H	4	OPVAL.	BYTE ARRAY(4) MEMBER(TYPE8STRUC)							629	631	
				632									
620	0000H	2	OPVAL.	BYTE ARRAY(2) MEMBER(TYPE7STRUC)							624	625	
678	0000H	2	OPVAL.	BYTE ARRAY(2) MEMBER(TYPE11STRUC)							683	686	
8	006BH	3	PREFIX.	BYTE ARRAY(3)			195	389					
23	0000H		PRINTM.	PROCEDURE EXTERNAL(12) STACK=0000H							43		
34	0000H		PRINTWORD.	PROCEDURE EXTERNAL(13) STACK=0000H							846	848	
525	0960H	28	PUTIMMED.	PROCEDURE STACK=000EH					549	555	828		
530	077CH	30	PUTIMMEDS.	PROCEDURE STACK=000EH					664				
393	072AH	15	PUTINSTR.	PROCEDURE STACK=0012H					400	472	548	575	
				582	591	719	733	814	820				
380	06E6H	21	PUTMEM.	PROCEDURE STACK=000AH					389	395	404	405	
				473	475	477	526	528	531	533	583	592	696
				734	749	751							
385	0701H	41	PUTPREFIX.	PROCEDURE STACK=000EH						394	881		
402	0749H	23	PUTWORD.	PROCEDURE STACK=0010H						684	720	759	760
				815	821								
493	0004H	1	REG.	BYTE PARAMETER AUTOMATIC						494	498		
479	0006H	1	REG.	BYTE PARAMETER AUTOMATIC						480	488		
6	0000H	16	REG16.	BYTE ARRAY(16) EXTERNAL(5)							167		
7	0005H	1	REG16TYPE.	BYTE MEMBER(OP2)				623	630	729	737		
7	0005H	1	REG16TYPE.	BYTE MEMBER(OP1)				616	741				
7	0003H	1	REG16TYPE.	BYTE MEMBER(OP)			169	255					
6	0000H	16	REG8.	BYTE ARRAY(16) EXTERNAL(6)						155			
7	0006H	1	REG8TYPE.	BYTE MEMBER(OP2)				495	601	799			
7	0006H	1	REG8TYPE.	BYTE MEMBER(OP1)									
7	0003H	1	REG8TYPE.	BYTE MEMBER(OP)			157	255					
9	0073H	1	REGBITS.	BYTE		328	473	488	498	507	512	605	611
				707	802	808							
226	03CEH	119	REGINDIRECT.	PROCEDURE STACK=0014H						273	302		
7	0001H	1	REGNUM.	BYTE MEMBER(OP2)				512	601	624	631	729	
				788									
127	0000H	1	REGNUM.	BYTE BASED(REGNUMPTR)						133			

COPYRIGHT © 1981, 1982
DIGITAL RESEARCH
P. O. BOX 579
PACIFIC GROVE, CA 93950

7	0001H	1	REGNUM	BYTE MEMBER(OP1)	507	741	827	
7	0001H	1	REGNUM	BYTE MEMBER(OP)	155	158	167	170 422 426
124	0004H	2	REGNUMPTR.	WORD PARAMETER AUTOMATIC			125	127 133
124	0008H	2	REGTADPTR.	WORD PARAMETER AUTOMATIC			125	126 131 136
126	0000H	2	REGWORD.	WORD BASED(REGTADPTR)			131	
8	0071H	1	REPFLAG.	BYTE	351	370	372	
10	000FH	1	RIGHIBRACKET . . .	BYTE DATA	232	249	295	
9	0074H	1	YNBITS	BYTE	422	426	430	437 443 448 452 456
				460 464 473				
8	006FH	1	SEGPREFIXFLAG. . .	BYTE	199	202	351	
6	0000H	8	SEGREG	BYTE ARRAY(8) EXTERNAL(4)			147	
7	0000H	1	SEGREGNUM.	BYTE MEMBER(OP2)	629	641	808	
7	0000H	1	SEGREGNUM.	BYTE MEMBER(OP1)	802			
7	0000H	1	SEGREGNUM.	BYTE MEMBER(OP)	147	201		
7	0004H	1	SEGREGTYPE	BYTE MEMBER(OP2)	628	641	805	
7	0004H	1	SEGREGTYPE	BYTE MEMBER(OP1)	799			
7	0004H	1	SEGREGTYPE	BYTE MEMBER(OP)	149	204	255	
7	000FH	2	SEGVALUE	WORD MEMBER(OP2)	760			
7	000FH	2	SEGVALUE	WORD MEMBER(OP1)				
7	000FH	2	SEGVALUE	WORD MEMBER(OP)	277			
397	0739H	16	SETINSTR	PROCEDURE STACK=0018H		624	629	631 683
				693 695 716 730 758 788		827		
557	09F0H	33	SETINSTRW.	PROCEDURE STACK=0004H		779	791	
411	0775H	37	SETHOD	PROCEDURE STACK=0002H		455	459	463
416	079AH	191	SETHODRK	PROCEDURE STACK=0006H		471		
193	0337H	24	SETPREFIX.	PROCEDURE STACK=0004H		201	366	373
198	034FH	47	SETSEGPREFIX . . .	PROCEDURE STACK=0012H		216	357	
407	0760H	21	SETWBIT.	PROCEDURE STACK=0002H		517	547	553 662
			SHL.	BUILTIN	111	201	473	629 749 827
			SHR.	BUILTIN	706			
4			STRINGTYPE	LITERALLY	104	179		
7	000EH	1	SUBTYPE.	BYTE MEMBER(OP2)	559	570	616	811
7	000EH	1	SUBTYPE.	BYTE MEMBER(OP1)	505	521	570	805 817
				825				
7	000EH	1	SUBTYPE.	BYTE MEMBER(OP)	160	172	235	240 245 297
				419				
2			TAB.	LITERALLY	10	120		
335	000CH	2	TABLEBASE.	WORD	335	338	340	345
124	0006H	1	TABLEN	BYTE PARAMETER AUTOMATIC			128	130
3	0000H	10	TADPTRS.	WORD ARRAY(5) EXTERNAL(0)			338	
739	0073H	1	TEMP	BYTE	742	744	749	750
4	0022H	32	TOKEN.	BYTE ARRAY(32)	4	90	91	120 211 232
				238 249 288 295 309 318		340	354	
4	0042H	1	TOKENLEN	BYTE	86	90	91	95 101 336 338 339
				340 342 345				
4	0043H	1	TOKENTYPE.	BYTE	97	103	104	179 209 266 280 291
4	0000H	2	TOKENVALUE	WORD	84	111	268	277 282 293
4	0022H	2	TOKENWORD.	WORD AT	131	181	183	
2			TRUE	LITERALLY	61	68	85	89 118 134 149
				150 157 159 162 169 171		174	186 202 207	
				212 228 234 283 285 298		343	352 365 372	
				375				
645	0004H	1	TYPE	BYTE PARAMETER AUTOMATIC		646	657	
572	0A53H	23	TYPE1.	PROCEDURE STACK=0022H		858		
675	0C7EH	11	TYPE10	PROCEDURE STACK=002EH		867		
679	0CB9H	68	TYPE11	PROCEDURE STACK=0024H		868		

COPYRIGHT © 1981, 1982

DIGITAL RESEARCH

P.O. Box 579

PACIFIC GROVE, CA 93950

SER =

678	001AH	4	TYPE11STRUC. . . .	STRUCTURE ARRAY(2) INITIAL	683	686
578	0A6AH	29	TYPE2.	PROCEDURE STACK=0022H	859	
587	0A87H	55	TYPE3.	PROCEDURE STACK=0022H	860	
596	0ABEH	82	TYPE4.	PROCEDURE STACK=0026H	861	
609	0B10H	23	TYPE5.	PROCEDURE STACK=0026H	862	
613	0B27H	37	TYPE6.	PROCEDURE STACK=0022H	863	
621	0B4CH	61	TYPE7.	PROCEDURE STACK=0026H	864	
620	000EH	4	TYPE7STRUC. . . .	STRUCTURE ARRAY(2) INITIAL	624	625
635	0BDFH	11	TYPE8.	PROCEDURE STACK=0028H	865	
639	0BEAH	30	TYPE8A.	PROCEDURE STACK=0028H	878	
627	0B89H	86	TYPE8OR8A. . . .	PROCEDURE STACK=0024H	637	643
634	0012H	8	TYPE8STRUC. . . .	STRUCTURE ARRAY(2) INITIAL	629	631 632
672	0C73H	11	TYPE9.	PROCEDURE STACK=002EH	866	
645	0C03H	107	TYPE9OR10. . . .	PROCEDURE STACK=002AH	673	676
754	0E2L	66	TYPECALLFJMPF. . .	PROCEDURE STACK=0024H	874	
701	0D05H	59	TYPEESC.	PROCEDURE STACK=0022H	870	
724	0D76H	82	TYPEIN.	PROCEDURE STACK=0022H	872	
688	0CCDH	56	TYPEINT.	PROCEDURE STACK=0022H	869	
796	0EECH	223	TYPEMOV.	PROCEDURE STACK=0028H	877	
738	0DC3H	102	TYPEOUT.	PROCEDURE STACK=0022H	873	
712	0D40H	54	TYPERETRETF. . . .	PROCEDURE STACK=001EH	871	
764	0E70H	66	TYPETEST.	PROCEDURE STACK=0022H	875	
784	0EB2H	58	TYPEXCHG.	PROCEDURE STACK=0022H	876	
334	05F3H	123	VALIDMNEMONIC. . .	PROCEDURE BYTE STACK=000EH	358	
7	0011H	2	VALUE.	WORD MEMBER(OP2)	526	528 531 533 536
					540	589 592 603 659 684 690 692 696 720
					731	734 759 815
7	0011H	2	VALUE.	WORD MEMBER(OP1)	704	706 707 743 751
					821	
7	0011H	2	VALUE.	WORD MEMBER(OP)	268	282 293 412 418
7	000CH	1	WBIT.	BYTE MEMBER(OP2)	47	49 408 511 560
					749	819
7	000CH	1	WBIT.	BYTE MEMBER(OP1)	47	49 408 527 536
					538	541 561 730 787 813 827
7	000CH	1	WBIT.	BYTE MEMBER(OP)	161	173 219 221 256 485
					437	
226	0004H	1	X.	BYTE PARAMETER AUTOMATIC	227	235 240 245

MODULE INFORMATION:

CODE AREA SIZE = 10C3H 4291D
 CONSTANT AREA SIZE = 0024H 36D
 VARIABLE AREA SIZE = 007FH 127D
 MAXIMUM STACK SIZE = 0036H 54D
 1130 LINES READ
 0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION

COPYRIGHT © 1981, 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____